Logan MacAskill

logan.macaskill.com | github.com/log2bits | (650) 237-9593
Los Altos, CA | linkedin.com/in/logan-macaskill-356190222 | logan@macaskill.com

About Me

Third-year CS student with two software engineering internships spent shipping and tuning production systems. I love problem solving, especially when I can take complete advantage of the hardware. I've worked on projects across realtime systems, graphics, AI, and data science in Rust, Java, C++, Python, and JavaScript. Currently working on my own custom Vulkan & C++ deferred renderer.

Experience

Paid Software Engineering Intern — **Flickr** — **Jun 2023 - Aug 2023**

- Cut compute cost 92% and raised throughput 39% on Wolverine, a Python AWS Lambda photo-repair service running over a library of ~10 billion photos, by tuning memory and thread allocation to the cheapest stable configuration.
- Profiled dozens of Lambda configurations on controlled batches of real jobs in Splunk, modeling cost per job, throughput, and tail latency to find the cost/performance point AWS docs do not surface.
- Recovered AWS's undocumented vCPU-to-memory scaling ratio by measuring CPU behavior across memory settings and fitting a linear regression, enabling accurate per-job cost estimates.

Paid Software Engineering Intern — **SmugMug** — **Jun 2022 - Aug 2022**

- Owned a user-facing gallery stats feature end to end (PHP, React, Next.js), adding daily refresh scheduling and clearer labels and empty states, shipped through PR review and QA.
- Wrote unit tests and backend changes supporting a release-critical PHP 8.1 upgrade, pairing with senior engineers to land it on schedule.
- Traced and fixed a broken checkout-flow link buried in a large monorepo, shipping the fix through a multi-stage review and QA pipeline.

Projects

ray-vox: Ray-traced Voxel Renderer — **Rust, WebGPU, Data Structures, Optimization** — **2026**

- Built the core sparse-voxel data structure for a from-scratch ray-traced renderer in Rust and WebGPU, storing an 84 MB scene in 16 MB, smaller than zstd's most aggressive setting (19 MB) while staying directly GPU-traversable.
- Eliminated all per-node child pointers using per-node bitmasks and bit-counting, and packed node offsets into a single 32-bit word, cutting memory while keeping ray traversal branch-light for GPU warps.
- Designed the format as a GPU acceleration structure with built-in level-of-detail and matching on-disk and in-GPU-memory layout, so uploads are near-zero-conversion copies.

Crowd Surfers: Real-time 3D Game — **Godot, Shaders, Lighting, Architecture** — **2025 - 2026**

- Led a mid-project migration from a faked-depth sprite system to true 3D with an angled orthographic camera, building a working prototype that convinced a 100-person student team to adopt the rewrite.
- Built a 3D occlusion-based transparency shader that fades buildings as the player skates behind them, using a camera-to-player frustum test plus dithered alpha to fit the alpha-cut asset pipeline.
- Added real-time shadows, dynamic lighting, and camera feel (speed-based FOV, screen shake, look-ahead), all smoothed with interpolation.

Real-time Dielectric Spectral Raymarcher — **GLSL, Spectral Rendering, Sampling, GPU** — **2026**

- Rendered a physically based dispersive diamond in a single real-time GLSL shader, trading a path tracer's temporal averaging for a 16x16 Bayer dither that schedules wavelengths across space, 65,536 distinct wavelengths resolved from two samples per pixel, with no denoiser and no accumulation buffer.
- Replaced raymarching with exact ray-plane intersection across the icosahedron's 20 faces (four golden-ratio directions, sign-flipped), and made light transport deterministic by peeling off exact Fresnel energy at every facet instead of stochastically sampling one path, noise-free at one sample, with surface normals free.
- Held real-time frame rates in a browser tab by cutting each ray once trapped energy fell below 4% (typically a handful of bounces against a 16-bounce cap) and keeping every pixel a self-contained shader invocation, no frame history, no neighbor reads, nothing to coordinate across the GPU.

Coaxial Swerve Drive — **Java, Control Theory, Computer Vision** — **2023 - 2024**

- Built a coaxial swerve drivetrain from scratch for FRC (custom CNC chassis, electronics, ~2,000 lines of Java) with no prior team experience, leading an off-season team to a working prototype in one month, ~4 months ahead of schedule.
- Ran per-module PID and feedforward at a 1 kHz control loop and derived current limits from robot mass and tire friction to maximize grip without slip or brownout, after self-teaching graduate-level control theory.
- Fused wheel encoders, gyro, and AprilTag vision through a Kalman-filter pose estimator holding sub-centimeter localization through wheel slip, and simulated the full robot (torque curves, inertia, battery, brownouts) on the exact production code.

Education

University of California — **Santa Cruz, CA** — **Expected Jun 2028**
B.S. in Computer Science | GPA: 3.6/4.0
Honors: Merit Scholarship, Dean's Honors
Relevant Coursework: Data Structures & Algorithms, Computer Architecture, Systems Programming in C, Linear Algebra, Vector Calculus, Probability & Statistics
Activities: UC Santa Cruz Game Design & Art Club

Skills

Languages: Rust, C++, Python, Java, TypeScript / JavaScript, PHP, GLSL / WGSL
Tools & Cloud: Git, Linux, AWS (EC2, Lambda, Graviton), Docker, OpenCV, Godot, Unity, React / Next.js
Concepts: Performance Optimization, GPU Compute, Control Theory, Computer Vision, Pose Estimation
Graphics & GPU: Vulkan, WebGPU, Ray Tracing, Real-Time Rendering, Rasterization, Shaders, Level-of-Detail

Future

Going to learn DirectX 12 (DX12) + HLSL as well as Unreal Engine after my Vulkan/C++ project